CSE 2017 Data Structures and Lab

Lecture #12: Graph 2

Eun Man Choi

Shortest-path problem

- There might be multiple paths from a source vertex to a destination vertex
- ***Shortest path***: the path whose total weight (i.e., sum of edge weights) is minimum

Austin→Houston→Atlanta→Washington: 1560 miles

Austin→Dallas→Denver→Atlanta→Washington: 2980 miles

Variants of Shortest Path

- Single-pair shortest path
  - Find a shortest path from u to v for given vertices u and v

- Single-source shortest paths
  - $G = (V, E) \Rightarrow$ find a shortest path from a given source vertex s to each vertex $v \in V$

Variants of Shortest Paths (cont'd)

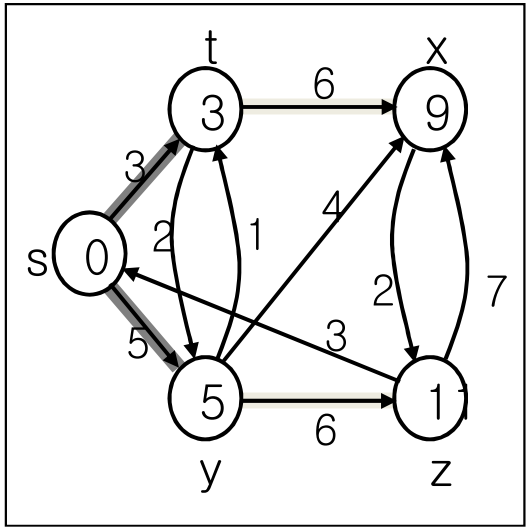
- Single-destination shortest paths
  - Find a shortest path to a given destination vertex t from each vertex v
  - Reversing the direction of each edge → single-source

- All-pairs shortest paths
  - Find a shortest path from u to v for every pair of vertices u and v

Notation

- **Weight of path** p = $\langle v_0, v_1, \ldots, v_k \rangle$

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$



- **Shortest-path weight** from s to **v**:

$$\delta(v) = \begin{cases} \min \; w(p) : s \overset{p}{\rightsquigarrow} v & \text{if there exists a path from s to v} \\ \infty & \text{otherwise} \end{cases}$$

Shortest-path algorithms

- Solving the shortest path problem in a brute-force manner requires enumerating all possible paths.
  - There are O(V!) paths between a pair of vertices in a acyclic graph containing V nodes.

- There are two algorithms
  - Dijkstra's algorithm
  - Bellman-Ford's algorithm(negative weights)

Shortest-path algorithms (cont'd)

- Dijkstra's algorithm is "greedy" algorithms!
  - Find a "globally" optimal solution by making "locally" optimum decisions.

- Both Dijkstra's algorithm is iterative:
  - Start with a shortest path estimate for every vertex:

    d[v]

  - Estimates are updated iteratively until convergence:

    $d[v] \rightarrow \delta(v)$

Shortest-path algorithms (cont'd)

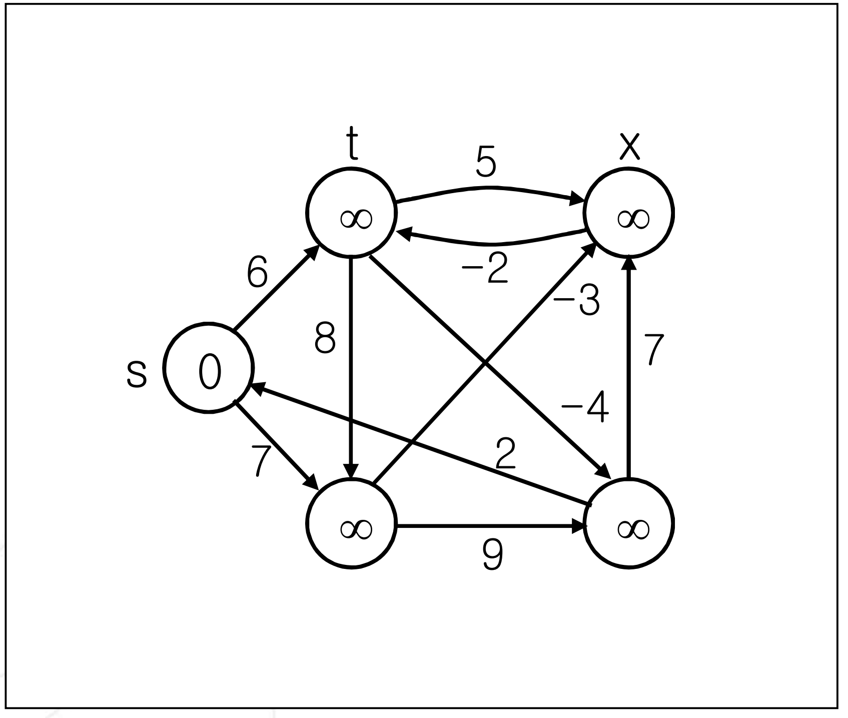
- **Two common steps:**

  (1) Initialization

  (2) Relaxation (i.e., update step)

Initialization Step

- Set d[s]=0 (i.e., source vertex)
- Set d[v]=∞ (i.e., large value) for $v \neq s$



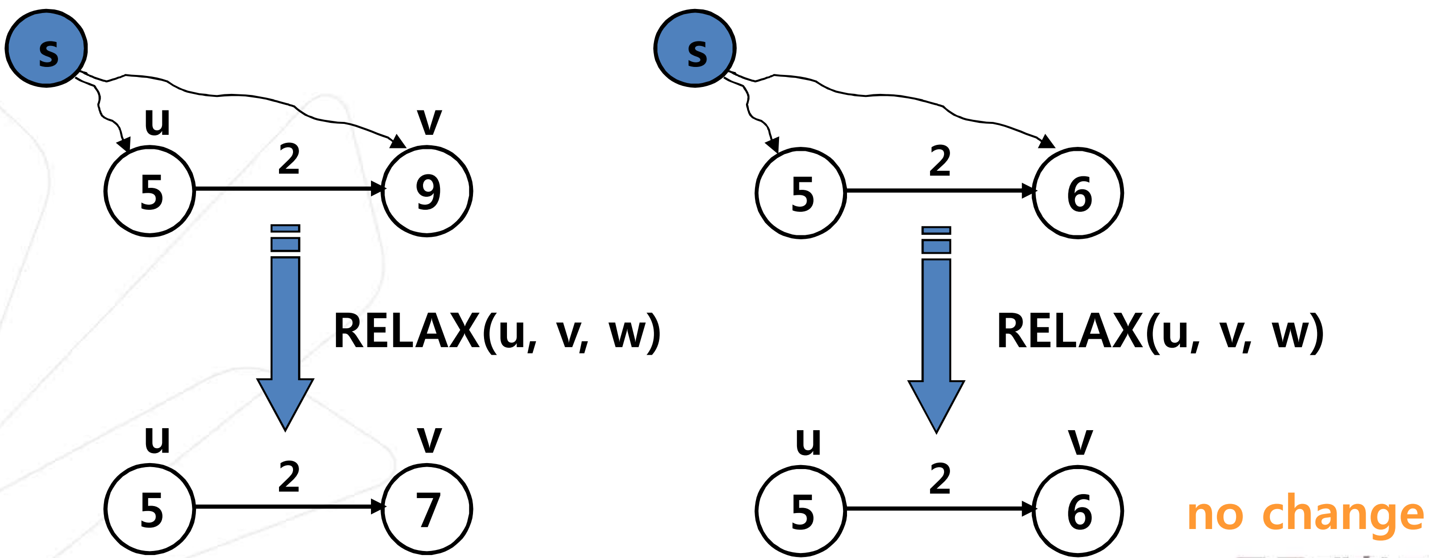
Relaxation Step

- Relaxing an edge (u, v) implies testing whether we can improve the shortest path to v found so far by going through u:

If d[v] > d[u] + w(u, v)

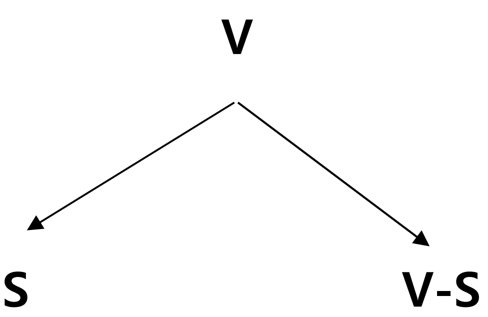
we can improve the shortest path to v

$\Rightarrow$ d[v]=d[u]+w(u,v)



Dijkstra's Algorithm (cont'd)

- At each iteration, it maintains two sets of vertices:



$d[v]=\delta(v)$

(estimates have converged to the shortest path solution)

$d[v]>\delta(v)$

(estimates have not converged yet)

Initially, S is empty

Dijkstra’s Algorithm (cont.)

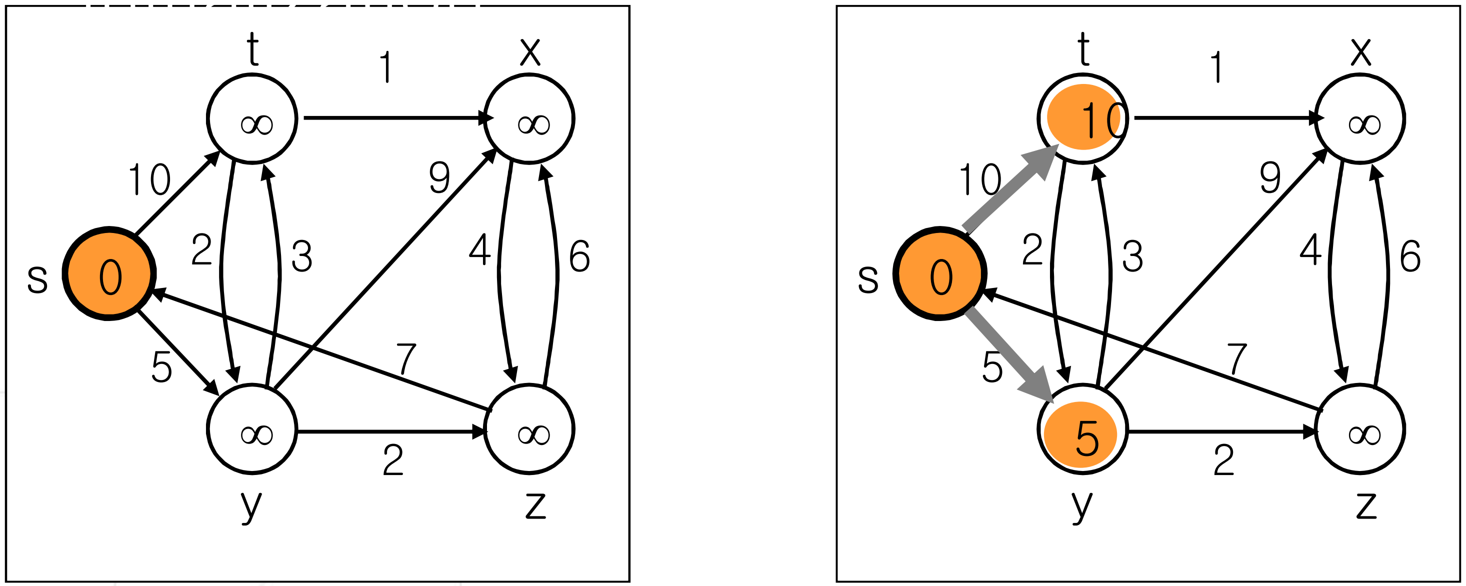
- Vertices in V–S reside in a min-priority queue Q
  - Priority of u determined by d[u]
  - The “highest” priority vertex will be the one having the smallest d[u] value.

Steps

1) Extract a vertex u from Q
2) Insert u to S
3) Relax all edges leaving u
4) Update Q

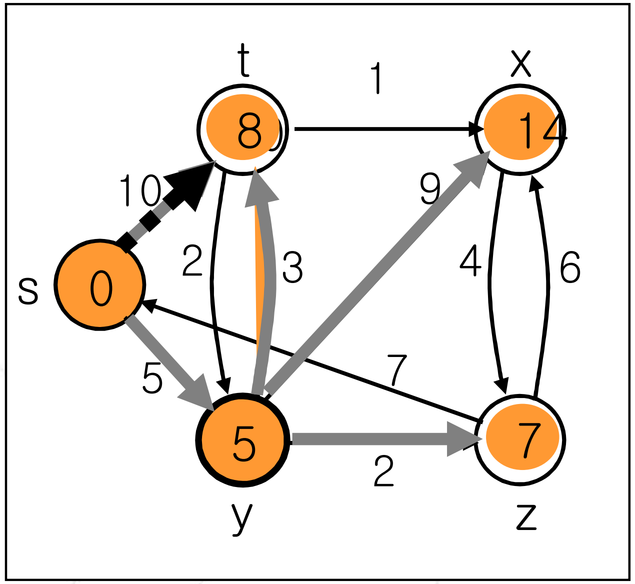
Dijkstra (G, w, s)

S=< > Q=<s,t,x,z,y> **S=<s> Q=<y,t,x,z>**

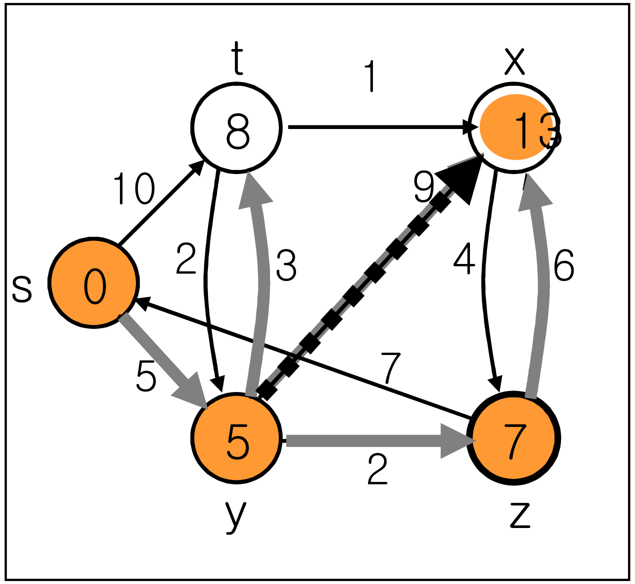


Example (cont.)

S=<s,y> Q=<z,t,x>

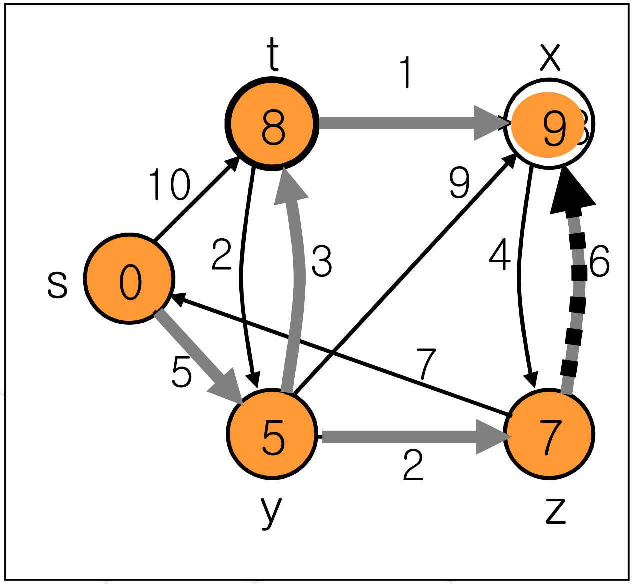


S=<s,y,z> Q=<t,x>

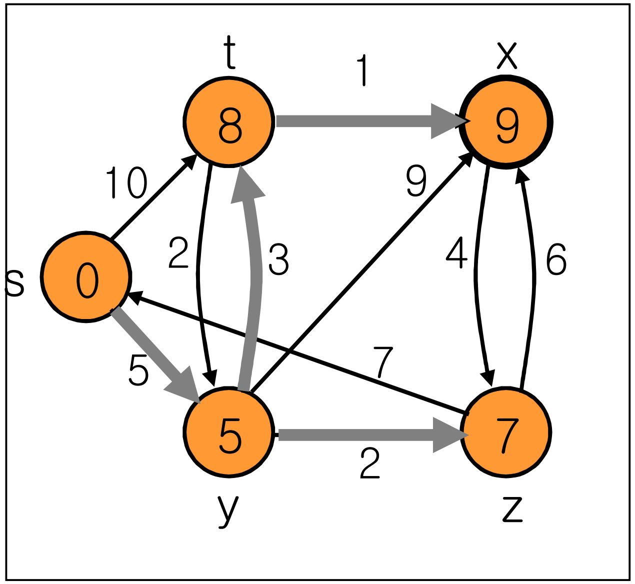


Example (cont.)

S=<s,y,z,t> Q=<x>



S=<s,y,z,t,x> Q=<>



<u>Note:</u> use back-pointers to recover the shortest path solutions

Dijkstra (G, w, s)

INITIALIZE-SINGLE-SOURCE(V, s) ← O(V)

S ← ∅

Q ← V[G] ← O(VlogV) – but O(V) is a tigther bound (build priority heap)

while Q ≠ ∅ ← O(V) times

do u ← EXTRACT-MIN(Q) ← O(logV)

S ← S ∪ {u}

for each vertex v ∈ *Adj*[u] ← $O(E_{vi})$

do RELAX(u, v, w)

Update Q (DECREASE_KEY) ← O(logV)

$O(E_{vi}\log V)$

Overall: $O(V+2V\log V+(E_{v1}+E_{v2}+\ldots)\log V) = O(V\log V+E\log V)=O(E\log V)$